NISTIR 5545

Neighbor Tables for Molecular Dynamics Simulations

R. D. Mountain

U.S. DEPARTMENT OF COMMERCE
Technology Administration
National Institute of Standards
and Technology
Gaithersburg, MD 20899

NIST

NISTIR 5545

Neighbor Tables for Molecular Dynamics Simulations

R. D. Mountain

U.S. DEPARTMENT OF COMMERCE
Technology Administration
National Institute of Standards
and Technology
Gaithersburg, MD 20899

December 1994

U.S. DEPARTMENT OF COMMERCE
Ronald H. Brown, Secretary

TECHNOLOGY ADMINISTRATION
Mary L. Good, Under Secretary for Technology

NATIONAL INSTITUTE OF STANDARDS
AND TECHNOLOGY
Arati Prabhakar, Director

Neighbor Tables for Molecular Dynamics Simulations

Raymond D. Mountain*
Thermophysics Division
Chemical Science and Technology Laboratory
National Institute of Standards and Technology
Gaithersburg, MD 20899

Abstract

Two methods for constructing a neighbor table for use in Molecular Dynamics simulations are discussed. A linked-list based method is shown to require time for generation of the table that is roughly proportional to the number of particles, N, in the system when the range of forces is small compared to the volume of the system. An alternative method that requires $\mathcal{O}(N^2)$ operations is also discussed and problems where it may be preferable to the linked-list method are mentioned. Source code listing for both methods is included in appendices.

Key words: linked list, molecular dynamics, neighbor table, particle simulation, sorting, source listing.

I. Introduction

In this note, we describe an important component, namely a neighbor table, of molecular dynamics and Monte Carlo codes used to simulate condensed matter at the atomic level. These simulations make it possible to calculate the thermodynamic and transport properties of liquids and solids in terms of the interactions between the atoms and molecules of the liquid or solid. These methods are, by now, fairly standard techniques in statistical physics and in condensed matter theory. A good introduction to the subject is found in the book by Allen and Tildesley.[1]

As computational capability increases, so does the complexity of the model systems studied using simulations. A goal of any simulation code development is that the computer time required per iteration scale with the size of the system. For this discussion, system size is measured by the number of particles, N, being simulated. Scalability with N is essential for the efficient use of the simulation method.

The use of a neighbor table is an important feature for improving the efficiency of molecular dynamics and Monte Carlo simulations of condensed matter at the atomic level. Typically,

* E-mail address: RMountain@nist.gov.

the system being simulated is subject to periodic boundary conditions and is confined to a cube with an edge length L. Forces are based on the minimum image convention and, since the interactions are usually assumed to be spherically symmetric, the particles outside a sphere of radius $R \leq L/2$ centered on a particle will not be considered. If the range of the interaction is less than $L/2$, even fewer pairs of interactions will be counted. The basic idea is that only particles that are within the range of the interaction should be included in the calculation of the forces between the particles. Particles with larger separations contribute nothing and calculating zero is a waste of time.

Verlet[2] introduced the use of a neighbor table in his early papers on the Lennard-Jones fluid. For the relatively small number of particles involved, say 500 or fewer, a "brute-force" approach to constructing the neighbor table was adequate. That is all pairs of particles were examined and those with separations less than some distance, d, were included in the table. Typically, d was taken to be a bit larger than the range of the interaction, R, so that it was necessary to update the neighbor table only occasionally, say every 10 time steps. This approach requires $\mathcal{O}(N^2)$ operations and becomes the dominant computational bottleneck for systems larger than a few hundred particles.

Since then, many alternative schemes have been proposed for constructing neighbor tables. The essential feature of these schemes is to partition the particles into adjacent groups and then to construct the table using these groups. There are several ways to do the partitioning. Two methods are discussed here.

We will discuss first a linked-list of cells scheme[3] that requires approximately $\mathcal{O}(N)$ operations to build the table and that can be readily adapted to a coarse-grained parallel computing environment. The basic idea used is a variation of an early suggestion by Quentrec and Brot.[4] It consists of partitioning the volume into cells with dimensions of at least twice the range of the table. The table is constructed by associating with each particle those particles located in the given particle's cell and in the near neighbor cells. A "spherical" neighbor table can be constructed by restricting the initial set of particles in the 27 cells to those within a spherical shell with a radius equal to the range of the neighbor table. This second step is necessary to remove the relatively large number of noninteracting pairs that are identified initially. A detailed description of how this can be realized is provided in Section II and FORTRAN77 code is listed in Appendix A.

The second scheme is based on work of Sullivan, et al.[5] Here, the partition is performed on the particles directly rather than on the volume containing the particles. It can be considered to be a variant of the "method of shadows". The scheme described in Section III is particularly useful for isolated systems, such as large clusters, where periodic boundary conditions are not appropriate and the density of particles is heterogeneous.[6] It is, however,

an inherently $\mathcal{O}(N^2)$ procedure. Under the test conditions described in Section III, it is more efficient than the linked-list method for systems of less than about $N = 3000$ particles. FORTRAN77 code is listed in Appendix B.

Source code for both methods is available from the author via an e-mail request.

II. The cell index method.

The system being simulated is located in a cube with edge length L. The minimum image convention and periodic boundary conditions are used so that surface effects are eliminated at the cost of introducing periodic images of the system. For the purposes of calculating the forces acting on a particle, that particle is assumed to be located at the center of a spherical region containing the particles that interact with it. The size of the spherical region is determined by the specific interaction rule and should not excede $L/2$. The neighbor table contains the labels of the particles in this spherical region, the neighbors, and only those particles are used to construct the forces. The task is thus to efficiently maintain this neighbor table as the simulation progresses. Particles will diffuse into and out of this region so it is necessary to update/rebuild this table. The cell index method is one way to do this.

The first step in the cell index method is to locate each particle in a cell embedded in the cube with side L. These cells are a set of n_c^3 cubes with side $l = L/n_c$. The dimension of these cells is at least twice the range desired for the neighbor table. That is $l \geq 2d$. In that way, all of the neighbors of particle j are to be found in the cell containing particle j or in one of the 26 cells (in three-dimensions) surrounding that cell, with due attention to the periodic boundary conditions. Of course, these 27 cells contain many particles that do not interact with particle j and must be removed from the final neighbor list.

The implementation of this scheme is illustrated in Appendix A. The subroutine `setup` produces a linked-list[3] of the 27 neighbor cells for each of the ${n_c}^3$ cells. First each cell is assigned a unique label stored in the array `nlabel`. Then the linked-list, `listcell`, is constructed. The periodic boundary conditions for the cells are taken into account in the `do 35`, `do 40`, and `do 45` loops. This subroutine is called only once during the simulation and the linked-list is passed in a common block.

The neighbor table itself is constructed in the subroutine `table`. Particles are assigned to a cell by dividing the x, y, and z coordinates by a length `xscale` and converting the results to integers. These integers identify the cell in which the particle is located. Once the particles have been placed in cells, as contained in the array `incell`, the neighbors of each particle are placed in a list, `lnbrs`, by copying the contents of `incell` according to `listcell`. This list is then reduced to the actual neighbor table, `nlist`, in the `do 60` loop. Note that it is necessary to impose periodic boundary conditions at this point.

This method requires that $n_c \geq 4$ in order to not be an $\mathcal{O}(N^2)$ process. In fact, the scalability with the number of particles of the time required to implement the process becomes evident only for n_c on the order of 6 or 7, at least for the example we now discuss. The actual efficiency also depends on how closely the side of the cell l matches twice the range of the neighbor table, d.

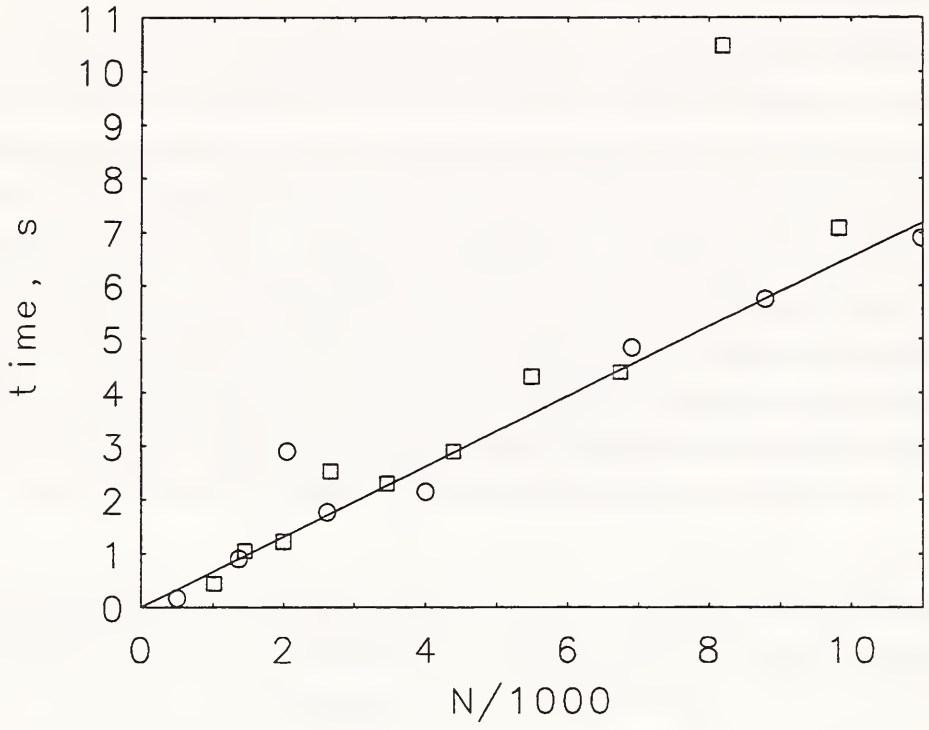

Fig. 1. The time, in seconds, required to generate a neighbor table for a range of system sizes. The circles are for particles on fcc sites and the squares are for particles on bcc sites. The straight line is intended only to guide the eye.

Timing tests have been performed for a collection of particles located on fcc or bcc lattice sites with a number density of 0.85 and a range for the neighbor table of 2.05. In the subroutine `table`, this would be identified as `range=2.05*2.05`. This density implies a unit cell dimension of 1.67 for the fcc lattice and of 1.33 for the bcc lattice. The tests were performed on a workstation and the approximate time required to construct the neighbor table as a function of the number of particles is shown in Fig. 1. The outliers are for $N = 2^{11} = 2048$ and $N = 2^{13} = 8192$ and are probably a consequence of specific hardware conditions as indicated in Fig. 2 where results for timings on a workstation and on a mainframe computer are compared. The actual times of course will depend on the computer system used, but the relative times indicate that the time needed to construct the neighbor table using this scheme approximately scales with system size for N up to 10,000 particles.

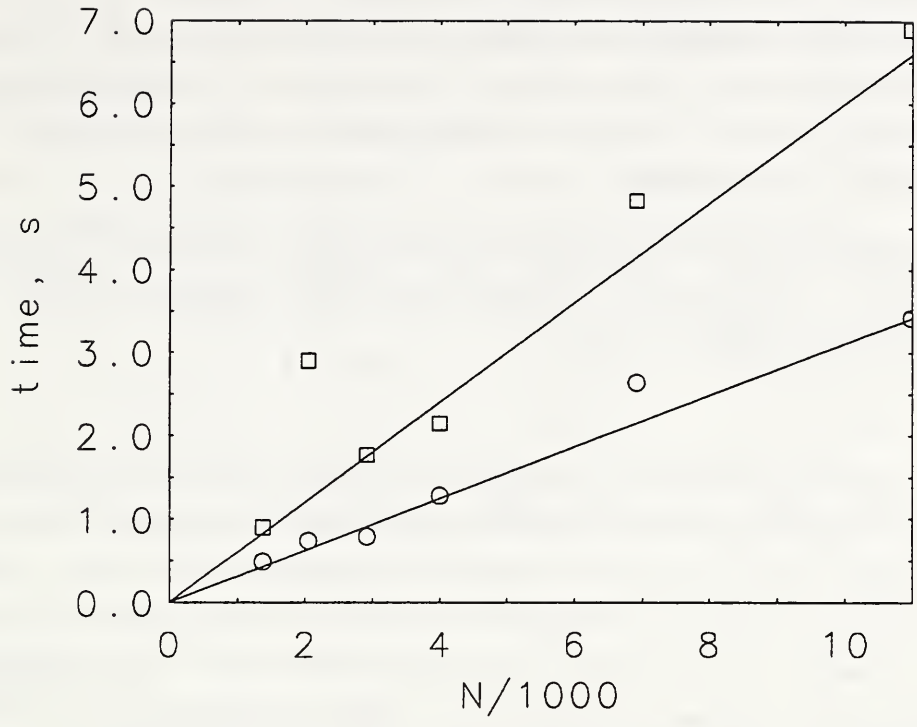

Fig. 2. A comparison of the time, in seconds, to generate a neighbor table using a workstation, squares, and using a mainframe computer, circles. Different hardware conditions on the mainframe suppress the outlier at $N = 2^{11}$ obtained using a workstation.

.

III. The method of shadows.

This approach is based on sorting.[5] First, one of the coordinates, say the x-coordinate, is sorted into increasing order. This makes it possible to identify all particles with x-displacements less than d from each particle. Then the y- and z- coordinates of this subset are examined to obtain the neighbors of a particle. Imagine a particle located at the center of a cube of side $2d$. All particles in that cube are neighbors. As with the linked-cell method, this set can be reduced to a spherical shell set of neighbors.

FORTRAN77 code to implement this scheme is listed in Appendix B. First, the `x` array is copied into the `sx` array and the particle labels are copied into the `locx` array. The subroutine `ssorti`, which is not listed, sorts `sx` into an increasing set and `locx` into the corresponding set of particle labels. The array `sx` is used to determine the positions of the left- and right-hand limits to the neighbors of each particle. These limits are recorded in the arrays `klower` and `kupper`. Next, this information is mapped back onto the original ordering in the `do 120` loop where the neighbors of particle `jj` are recorded in array `nbr`. Then the y- and z-distances are examined in the `do 150` loop where the neighbors of

particle `jj` are recorded in the array `nlist` and the number of neighbors is put in array `nj`. Finally, a spherical shell neighbor list is produced in the `do 600` loop. Because the number of entries in the array `nbr` is proportional to the number of particles in the system, this method is inherently an $\mathcal{O}(N^2)$ method. In practice, it can be faster than the cell index method under some circumstances.

This version does *not* utilize periodic boundary conditions as it was intended for the simulation of isolated clusters. A method for incorporating periodic boundary conditions into the procedure is discussed by Sullivan, et al.[5]

IV. Discussion.

A cell index method, that uses a linked-list of neighbor cells when constructing the neighbor table, has been shown to have the property that the time required to construct the neighbor table scales approximately linearly with the number of particles in the system. Since each particle has its own list of neighbors determined in the `do 50` loop, this method should readily lend itself to a parallel, coarse-grained computing environment. The execution time to build the table for a given number of particles, `np`, would be reduced by the number of processors, `nproc` available provided the communication overheads are not too large.[7]

These conclusions are based on the condition that the range of the neighbor table is small compared with the size of the system so that a large number of cells can be utilized. This is the case for short-ranged interactions such as the Stillinger-Weber potential for silicon.[8]. It does not apply to ionic or dipolar systems where the interactions are long ranged.[1] In such situations, the Ewald summation approach is used and the configuration part of the interaction extends to one-half the size of the simulation box. Clearly, some other strategy is needed for the efficient simulation of these system.

References

1. M. P. Allen and D. J. Tildesley, *Computer Simulation of Liquids*, (Carenndon Press, Oxford, 1987).
2. L. Verlet, Phys. Rev. **159**, 98 (1967).
3. G. S. Grest, B. Dünweg, and K. Kremer, Comput. Phys. Comm. **55**, 269 (1989).
4. B. Quentrec and C. Brot, J. Comput. Phys. **13**, 430 (1975).
5. F. Sullivan, R. D. Mountain, and J. O'Connell, J. Comput. Phys. **61**, 138 (1985).
6. E. Blaisten-Barojas and M. R. Zachariah, Phys. Rev. B **45**, 4403 (1992).
7. W. J. Camp and S. J. Plimpton, Proceedings of the 1993 Simulation Multiconference on High Performance Computing Symposium, A. M Tentner, ed. (The Society for Computer Simulation, San Diego, CA, 1993) pp. 127-141.
8. F. S. Stillinger and T. A. Weber, Phys. Rev. B **31**, 5262 (1985).

Appendix A.

This is a FORTRAN77 listing of the routines needed to implement a linked-list based neighbor table. First the cell-lists are generated at the beginning of a run. Note that the periodic boundary conditions for the cells are included in the lists. The subroutine `setup` is called only at the beginning of the simulation.

```
      subroutine setup
      parameter(np=512,nc=4,np3=nc*nc*nc)
      dimension nlabel(nc,nc,nc),listcell(np3,27)
      common /cset/ nlabel,listcell
c
      nl=0
      do 30 n1=1,nc
        do 25 n2=1,nc
          do 20 n3=1,nc
             nl=nl+1
             nlabel(n1,n2,n3)=nl
20        continue
25      continue
30    continue
      nl=0
      do 60 n1=1,nc
        do 55 n2=1,nc
          do 50 n3=1,nc
            nl=nl+1
            nll=0
            do 45 nx=n1-1,n1+1
               if(nx.lt.1) then
                  nxx=nc
               else if(nx.gt.nc) then
                  nxx=1
               else
                  nxx=nx
```

```
                end if
               do 40 ny=n2-1,n2+1
                  if(ny.lt.1) then
                     nyy=nc
                  else if(ny.gt.nc) then
                     nyy=1
                  else
                     nyy=ny
                  end if
                 do 35 nz=n3-1,n3+1
                    if(nz.lt.1) then
                       nzz=nc
                    else if(nz.gt.nc) then
                       nzz=1
                    else
                       nzz=nz
                    end if
                   npp=nlabel(nxx,nyy,nzz)
                   nll=nll+1
                   listcell(nl,nll)=npp
35               continue
40             continue
45           continue
50         continue
55       continue
60     continue
       return
       end
```

The subroutine `table` is called every few, say 6-10, time steps.

```
       subroutine table
       parameter (np=512,nc=4,np3=nc*nc*nc)
       dimension x(np),y(np),z(np),mj(np),nlist(40,np),lnbrs(np,400)
       dimension nlabel(nc,nc,nc),listcell(np3,27),lmbn(np)
       dimension incell(np3,40),lj(np3),nj(np)
       common /cset/ nlabel,listcell
       common /cord/ x,y,z,npart,xmax,
      x     xmax2,ymax,ymax2,zmax,zmax2,range
       common /clst/ mj,nlist
c
       do 10 j=1,np
         mj(j)=0
10     continue
       do 12 nn=1,np3
         lj(nn)=0
12     continue
       xscale=xmax/(1.*nc)+.05
       do 20 j=1,np
         ix=x(j)/xscale+1
         iy=y(j)/xscale+1
         iz=z(j)/xscale+1
         nn=nlabel(ix,iy,iz)
         nj(j)=nn
         lj(nn)=lj(nn)+1
         incell(nn,lj(nn))=j
```

```
20    continue
c     At this point, we have determined the contents of each cell
      do 50 j=1,np
        nn=nj(j)
        jk=0
        do 45 lk=1,27
          ik=listcell(nn,lk)
          do 40 nk=1,lj(ik)
            k=incell(ik,nk)
            if(k.ne.j) then
               jk=jk+1
               lnbrs(j,jk)=k
            end if
40        continue
45      continue
       lmbn(j)=jk
50    continue
c     Next, convert this to lists of particles within a shell
      do 60 j=1,np
        l=1
        do 55 kk=1,lmbn(j)
          k=lnbrs(j,l)
            l=l+1
          xx=abs(x(j)-x(k))
          yy=abs(y(j)-y(k))
          zz=abs(z(j)-z(k))
           if(xx.gt.xmax2) xx=xx-xmax
           if(yy.gt.ymax2) yy=yy-ymax
           if(zz.gt.zmax2) zz=zz-zmax
          rr=xx*xx+yy*yy+zz*zz
          if(rr.gt.range) go to 55
            mj(j)=mj(j)+1
            nlist(mj(j),j)=k
55        continue
60    continue
      return
      end
```

Appendix B.

This is FORTRAN77 source for the generation of a neighbor table using the scheme described in Section III. First one of the coordinates, the x-coordinate, is sorted into increasing order as are the particle labels. That is `sx(j)` and `locx(j)` are the x-coordinates in increasing order and the corresponding particle labels. The routine `ssorti(sx,locx,np,2)` orders the arrays `sx` and `locx` for `np` particles. This subroutine is a modified version of the *cmlib* routine `ssort`.

```
      do 2 j=1,np
        sx(j)=x(j)
        locx(j)=j
2     continue
      call ssorti(sx,locx,np,2)
      call table
```

Here is the source code for `table`.

```
      subroutine table
      parameter(np=6912)
      dimension x(np),klower(np),kupper(np)
      dimension y(np),z(np),sx(np),locx(np)
      dimension nlist(np,500),nbr(np),nj(np)
      dimension mlist(np,80),jmax(np)
      common /cord/ x,y,z,sx,locx,range
      do 50 j=1,np
       if(j.eq.1) then
        do 10 k=1,np
          d=sx(k)-sx(j)
          if(d.gt.-range) then
             klower(j)=k
             go to 15
          end if
10       continue
15       continue
         do 20 k1=k,np
           d=sx(k1)-sx(j)
           if(d.gt.range) then
             kupper(j)=k1
             go to 25
           end if
           kupper(j)=np
20       continue
25       continue
       else
         do 30 k=klower(j-1),np
           d=sx(k)-sx(j)
           if(d.gt.-range) then
              klower(j)=k
              go to 35
           end if
30       continue
35       continue
         do 40 k1=k,np
           d=sx(k1)-sx(j)
           if(d.gt.range) then
             kupper(j)=k1
             go to 45
           end if
           kupper(j)=np
40       continue
45     continue
       end if
50    continue
c
c     Map back to unsorted labeling and determine the neighbors.
c
      do 200 j=1,np
        jj=locx(j)
c       jj is the label in the original arrangement
        l=0
        do 120 k=klower(j),kupper(j)
```

```
         l=l+1
         nbr(l)=locx(k)
120     continue
c        nbr is the list of neighbors in the "slab"
         nj(jj)=l
c     Examine the y- and z- distances
         ll=0
        do 150 k=1,nj(jj)
         dy=abs(y(jj)-y(nbr(k)))
          if(dy.le.range) then
            dz=abs(z(jj)-z(nbr(k)))
             if(dz.le.range) then
               ll=ll+1
               nlist(jj,ll)=nbr(k)
             end if
           end if
150      continue
        nj(jj)=ll
200   continue
c
c     nlist(j,l) contains the neighbors of j in the "cube"
c     nj(j) is the number of neighbors of j.
c
        r2=range*range
      do 600 j=1,np
       l=0
       do 550 jj=1,nj(j)
         k=nlist(j,jj)
         dx=x(j)-x(k)
         dy=y(j)-y(k)
         dz=z(j)-z(k)
         rr=dx*dx+dy*dy+dz*dz
         if(rr.le.r2) then
           l=l+1
           mlist(j,l)=k
         end if
550    continue
       jmax(j)=l
      if(jmax(j).gt.60) then
       write(6,55) j,jmax(j)
55     format('trouble!',2i8)
       stop
      end if
600   continue
      return
      end
```